LIST OF FIGURES

ABSTRACT

Handwriting recognition is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "off line" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition.

Offline Handwritten Text Recognition (HTR) systems transcribe text contained in scanned images into digital text. We will build a Neural Network (NN) which is trained on word-images.

First, the handwriting to be recognized is digitized through scanners or cameras. Second, the image of the document is segmented into lines, words, and individual characters. Third, each character is recognized using OCR techniques. Finally, errors are corrected using lexicons or spelling checkers.

TABLE OF CONTENTS

INTRODUCTION

Neural networks are used as a method of deep learning, one of the many subfields of Artificial Intelligence. They were first proposed around 70 years ago as an attempt at simulating the way the human brain works, though in a much more simplified form. Individual ‘neurons’ are connected in layers, with weights assigned to determine how the neuron responds when signals are propagated through the network to simulate, and therefore the complexity of learning they could achieve. But in recent years, due to advancements in hardware development, we have been able to build very deep networks and train them on enormous datasets to achieve breakthroughs in machine intelligence.

These breakthroughs have allowed machines to match and exceed the capabilities of humans in performing certain tasks. One such task is object recognition. Although machines have historically been unable to match human vision, recent advances in deep learning have made it possible to build neural networks which can recognize objects, faces, text, and even emotions.

In this project, we will implement a small subsection of object recognition-digit recognition. Using TensorFling research, we will take images of handwritten data and build and train a neural network to recognize and predict the correct label for the data displayed.

While we won’t need prior experience in practical deep learning or TensorFlow to follow along with this project, we’ll assume some familiarity with machine learning terms and concepts such as training and testing, features and labels, optimization, and evaluation.

EXISTING WORK

1. Healthcare and Pharmaceuticals

Patient prescription digitization is a major pain point in the healthcare/pharmaceutical industry. For example, Roche is handling millions of petabytes of medical PDFs daily. Another area where handwritten text detection has a key impact is patient enrollment and form digitization. By adding handwriting recognition to their toolkit of services, hospitals/pharmaceuticals can significantly improve user experience.

2. Insurance

A large insurance industry receives more than 20 million documents a day and a delay in processing the claim can impact the company terribly. The claims document can contain various different handwriting styles and pure manual automation of processing claims is going to completely slow down the pipeline

3. **Banking**

People write cheques on a regular basis and cheques still play a significant role in most non-cash transactions. In many developing countries, the present cheque processing procedure requires a bank employee to read and manually enter the information present on a cheque and also verify the entries like signature and date. As a large number of cheques have to be processed every day in a bank a handwriting text recognition system can save costs and hours of human work

4. **Online Libraries**

Huge amounts of historical knowledge is being digitized by uploading the image scans for access to the entire world. But this effort is not very useful until the text in the images can be identified which can be indexed, queried and browsed. Handwriting recognition plays a key role in bringing alive the medieval and 20th-century documents, postcards, research studies etc.

LIMITATIONS

- Huge variability and ambiguity of strokes from person to person
- The handwriting style of an individual person also varies from time to time and is inconsistent
- Poor quality of the source document/image due to degradation over time
- Text in printed documents sit in a straight line whereas humans need not write a line of text in a straight line on white paper
- Cursive handwriting makes separation and recognition of characters challenging
- Text in handwriting can have variable rotation to the right which is in contrast to the printed text where all the text sits up straight
- Collecting a well-labelled dataset to learn is not cheap compared to synthetic data

PROPOSED WORK AND METHODOLOGY

Once we'll be done with installing Python 3, TensorFlow 1.3, NumPy and OpenCV, we need to start working on the project's layout and on the following domains which will be used

- **CNN (convolutional neural network)**

The input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. Each layer consists of three operations.

- **RNN (recurrent neural network)**

the feature sequence contains 256 features per time step, the RNN propagates relevant information through this sequence.

- **CTC (Connectionist Temporal Classification)**

while training the NN, the CTC is given the RNN output matrix and the ground truth text and it computes the **loss value**. While inferring, the CTC is only given the matrix and it decodes it into the **final text**

For implementation, we will use the following modules

- **SamplePreprocessor.py:** prepares the images from the IAM dataset for the NN
- **DataLoader.py**: reads samples, put them into batches and provides an iterator-interface to go through the data
- **Model.py**: creates the model as described above, loads and saves models, manages the TF sessions and provides an interface for training and inference
- **main.py**: puts all previously mentioned modules together

Building Python Deep Learning Project on Handwriting Recognition

1. Import the libraries and load the dataset
2. Preprocess the data
3. Create the model
4. Train the model
5. Evaluate the model
6. Create GUI to predict digits

Methodology:

Given an address block in PGM format, this project proceeds in the following steps to generate the desired output.

The steps are described below:

- **Building feature-graph database**: All the capital letters are broken up into features and a feature-graph is constructed and put into the database. The details of features and feature-graphs are explained in the following section. Note that corresponding to one alphabet, there may be more than one feature-graphs depending on the most prevalent ways in which people write that character.
- **Removing Noise:** Given the address block as an image, the first step is to remove the unnecessary noise in the image. Even though it is a simple step, it is a very necessary step that facilitates the later processings. The technique used for removing noise is the one suggested by [Lee/Gomes:1999].
- **Thinning:** The aim of thinning is to make all written characters (i.e. the black part of the image) one-pixel wide. This is a vital step that forms the basis for recognition as well as beautification. For the purpose of thinning, we first decided to use the algorithm provided by [Datta/Parui:1994]. This algorithm makes sure that continuity is maintained while the image is being thinned. But since this algorithm is not primarily meant for characters i.e. irregularly shaped and irregularly thick objects, it created some weird results for our case. Thus, we created a partial thinning algorithm that finds out the average width of the characters in the image and if this width is more than 3, the partial thinning algorithm thins all the characters depending on their width. After the partial thinning algorithm, we apply the thinning algorithm suggested by [Datta/Parui:1994].

- **Slant Estimation:** After getting a thinned image, it is scanned from left to right and all the vertical/tilted lines in the whole image are detected. The average of the slants of these lines gives us the slant of the image.

- **Slant Correction:** Once the slant has been detected, slant correction is applied to the original image. The corrected image is thinned again.
- **Separating Lines:** In this step, the address block is divided into lines and each line is looked at separately.
- **Separating Characters:** Studying a large number of samples, we observed that capital letters are generally written with different characters some distance apart. Hence, we decided to break up a line into characters so as to make the feature extraction and matching simpler.
- **Feature Extraction:** This is the most interesting step. Here, each row of the image is scanned from left to right and top to bottom to figure out the positions of black pixels. Depending on these positions, the features horizontal line, vertical line, left slant, right slant, upper curve, lower curve, left curve and right curve are intelligently detected keeping in mind the variance in these features that may arise due to differences in handwriting. In some cases, when one feature can be mistaken for another, precaution is taken to detect the features in a pre-defined order and mark the visited black pixels so that confusions are avoided.
- **Matching:** This is the recognition step. In feature extraction, we work to get the set of features between two white columns. This set of features may or may not be of a single character. Once the features are detected, the following continuous matching algorithm is applied to know the set of possibilities that the set of features may denote.
- **Continuous Matching:** In the first step, the set of feature graphs in the database which have the first feature (in the set of features detected) is chosen. We call this the current set. In subsequent steps, the rest of the features are looked for in the current set and the feature graphs in the current set that does not have that feature are discarded. In this way, the current set gets smaller at each step. Also, if, at any stage, all of the features in a feature-graph have been found, then it is checked for relative directions of the features. If the directions match, the character corresponding to this feature-graph is added to the possibility and the Continuous Matching algorithm is called recursively with the rest of the features. Note that, due to recursive calls and backtracking, this algorithm is able to generate all the correct possibilities for a set of features.

NOVELTY

We'll be using the concepts of Deep Learning and neural network, the convolutional neural network (CNN) and the recurrent neural network (RNN) in our project. CNN's are great at interpreting visual data and give better results with accuracy up to 98.92% whereas RNNs are used to analyze sequential input and are not limited by length. In this project, we tend to try and demonstrate how by using NumPy and python we will produce a neural network that would not only recognize digits but also alphabets.

REAL TIME USAGE

The 21st century is digitally connected. We no longer send handwritten letters and rarely use printed texts for the simple reason that we became too dependent on our computers to process data and make life easier. These reasons made us need to find a way to digitize physical papers in a way to be electronically edited, manipulated, searched, managed, stored, and especially interpreted by machines. Optical Character Recognition made it possible to convert text captured in images of typed, handwritten, or printed text into digitized and usable machine-encoded text.

In the same way, our project will be able to scan texts and give the live results which it will later store in it's memory. It will be based on concepts of OCR which is a field of research in Artificial Intelligence and Computer Vision that consists in extracting text from images.

HARDWARE AND SOFTWARE REQUIREMENTS

Hardware:

A decent working laptop for running the model and mobile phone for clicking and transferring the photo

Software:

- KERAS
- PYTHON
- MNIST DATASET (Modified National Institute of Standards and Technology database)
- TKINTER LIBRARY
- CNN MODEL
- TENSORFLOW
- NUMPY
- PILLOW

Overall System Architecture Diagram

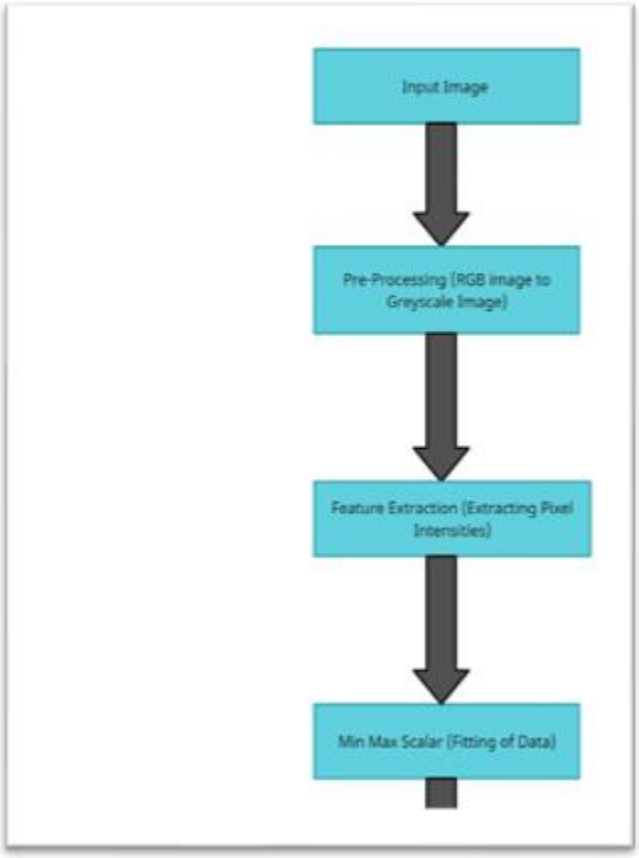


Fig.1

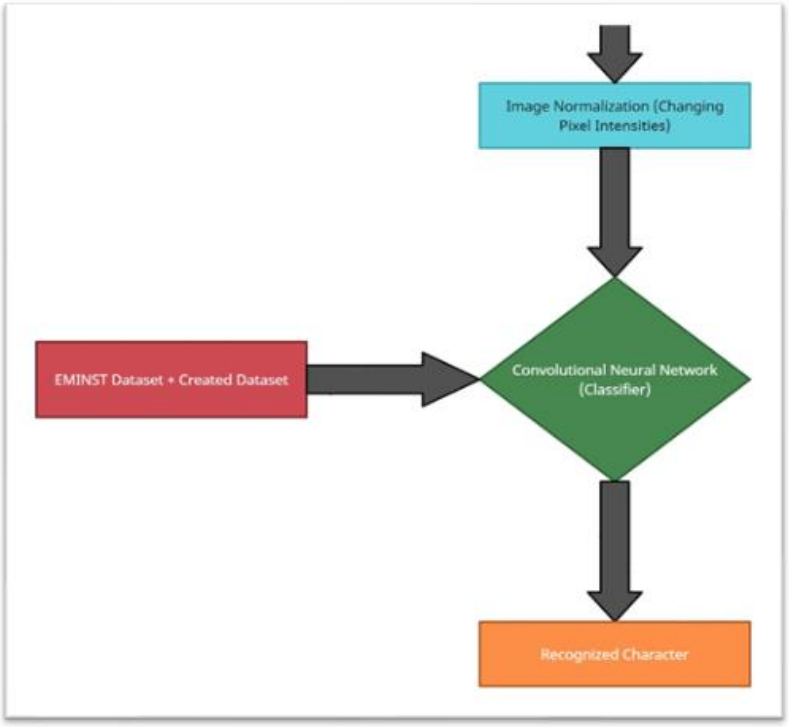


Fig.2

First we run the code we get a display screen drawing screen in which we have to input the digit you need to recognize. After that the digit which we had drawn will pre-process it. During this pre-processing the image will start converting from RGB to Grayscale image for better recognition and enhancement of pixels. Now these pixels intensities are extracted and they are level by min max scalar to fit the data to recognise the digit from the dataset. Then these pixels change their intensities as per the dataset pixels so that they can predict the digits from EMNIST dataset. Then the output is shown.

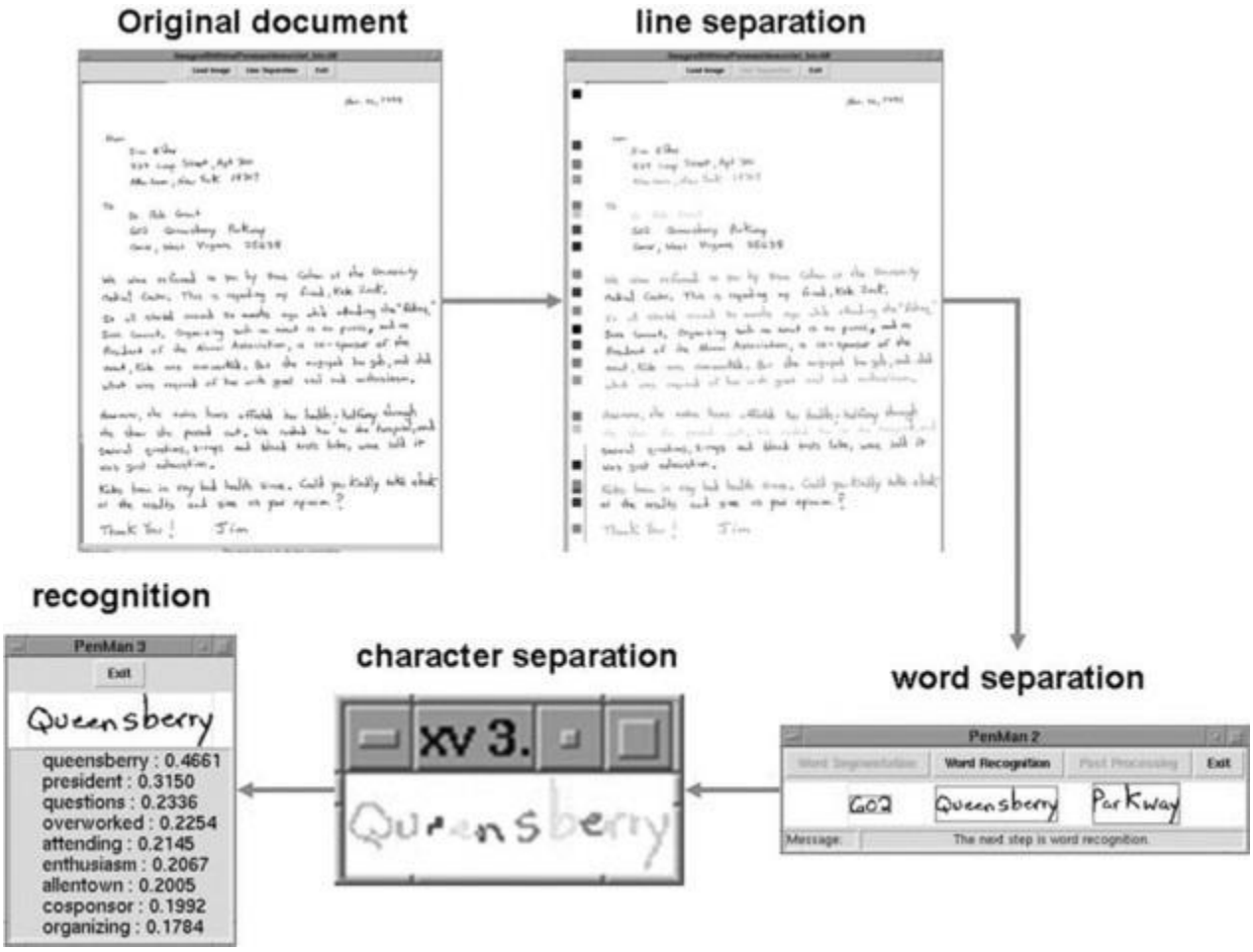
LITERATURE REVIEW

•The earliest Optical Character Recognition systems were not computers but mechanical which were very slow and very less accurate.In 1951, Sheppard invented a robot GISMO which used to read musical notation and could copy a typewritten page.

•From the past 30 years research has been done in this field and has led to emergence of document image analysis, multilingual, handwritten and omni-font OCRs. The current OCR research is being done on improving accuracy and speed for diverse style documents

•New tools not only observe the text but make changes like line spacing and word separation. Software like google vision, AWS Textract and azure API, systems consist of four processes which are acquisition, segmentation, recognition and postprocessing.

•Tesseract OCR and SemaMediaData are some offline tools which are less accurate because only spatial information is available.



MODEL DESCRIPTION

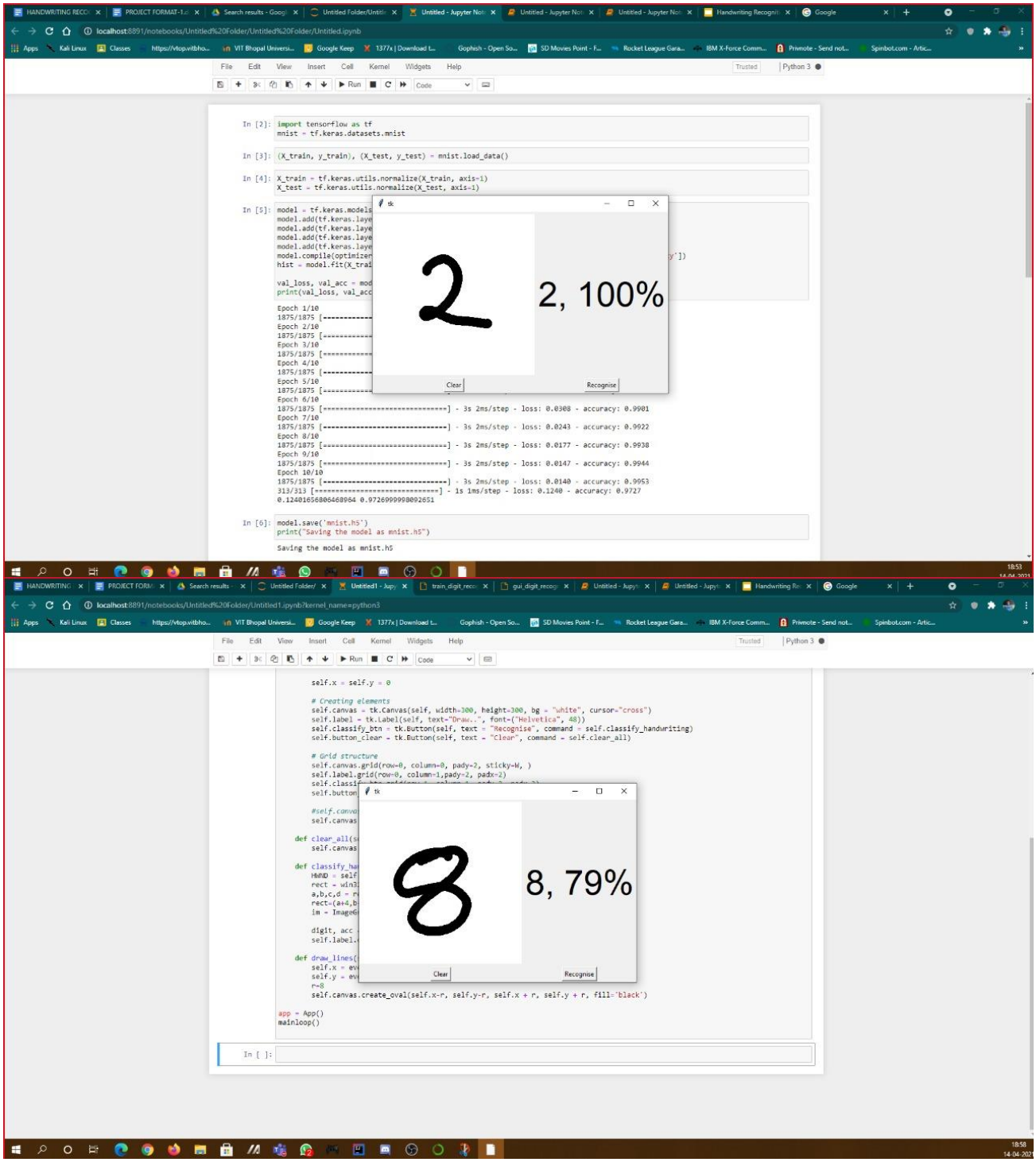
We use a NN for our task. It consists of a convolutional neural network (CNN) layer, recurrent neural network (RNN) layers, and a final Connectionist Temporal Classification (CTC) layer. In this project, we've taken 5 CNN (feature extraction) and a pair of RNN layers and a CTC layer (calculate the loss). First, we've to preprocess the pictures in order to reduce the noise. We can also view the NN in an exceedingly more formal way as a function which maps a picture (or matrix) M of size W×H to a personality sequence (c1, c2, …) with a length between 0 and L. As you'll see, the text is recognized on character-level, therefore words or texts not contained within the training data are recognized too (as long because the individual characters get correctly classified).

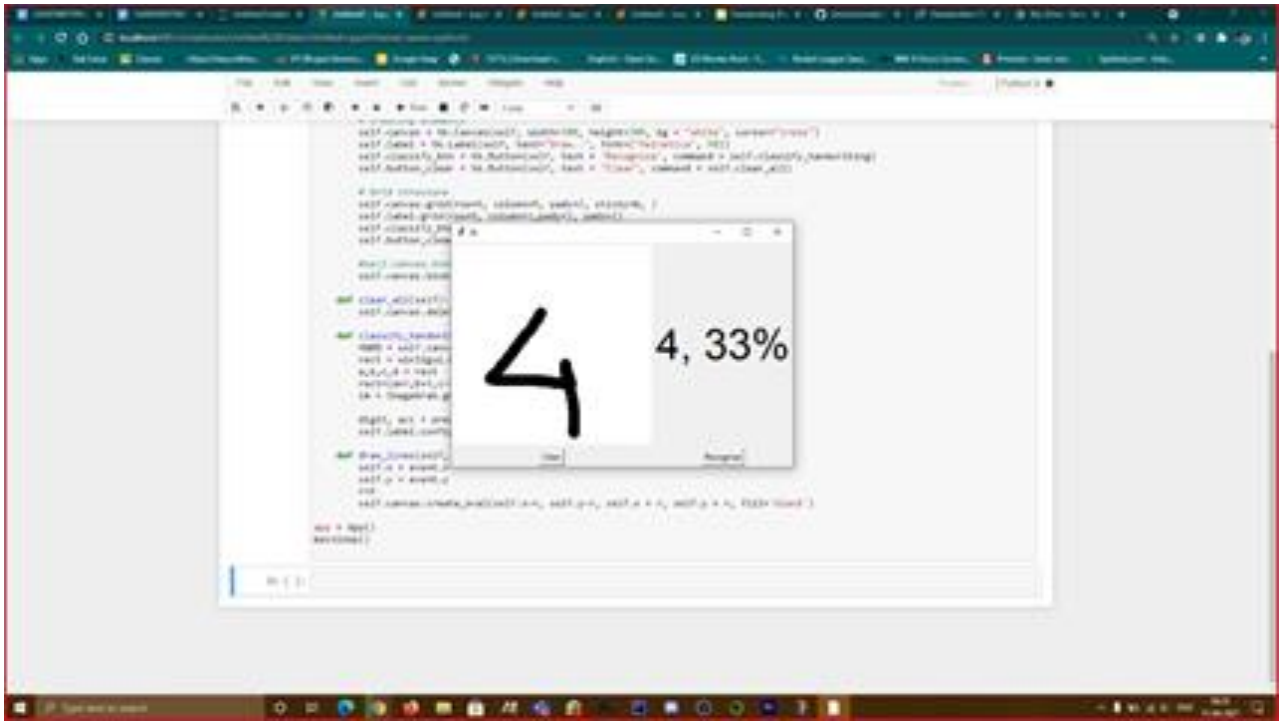
MODULE WORKFLOW EXPLANATION

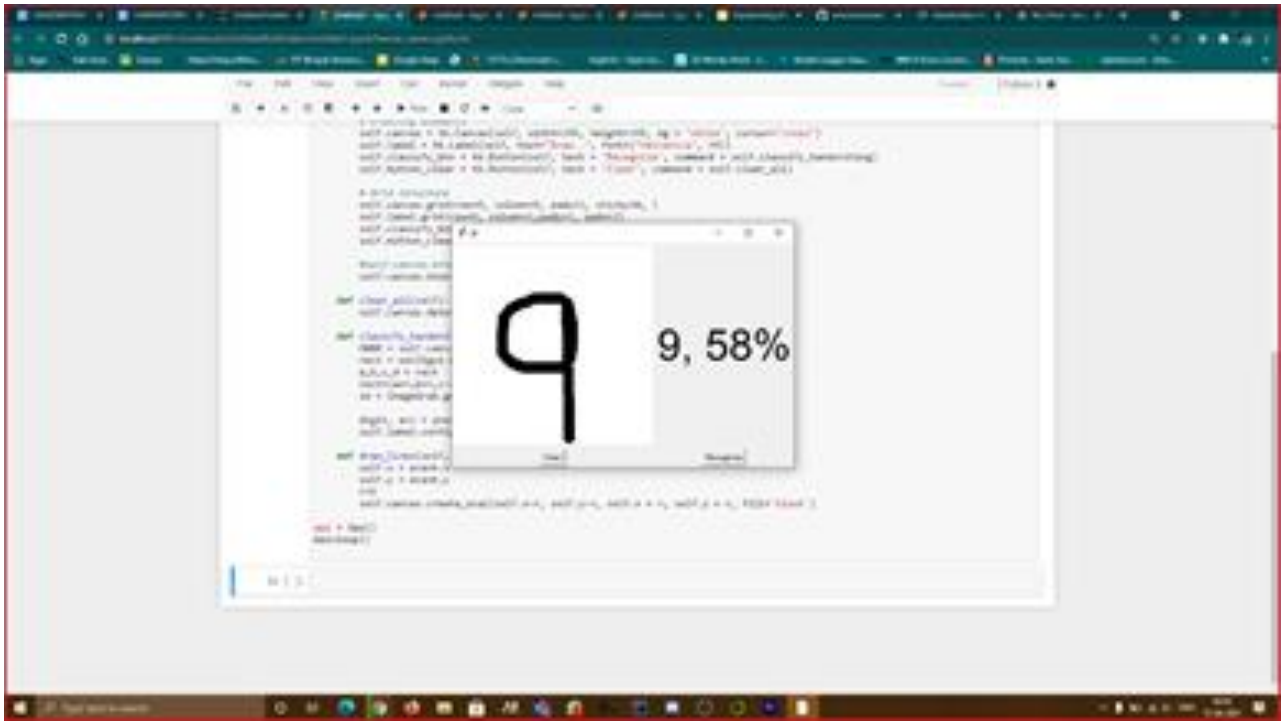
In Digit Recognition
•We first imported modules TensorFlow and KERAS.
•Then we train models, but we split the models in two parts that are x and y.
•Then we will load the dataset then the model starts training while training it also checks the accuracy and loss on samples.
•We will save the weights of the model which we trained upon.
•We will load the model and use it for testing.

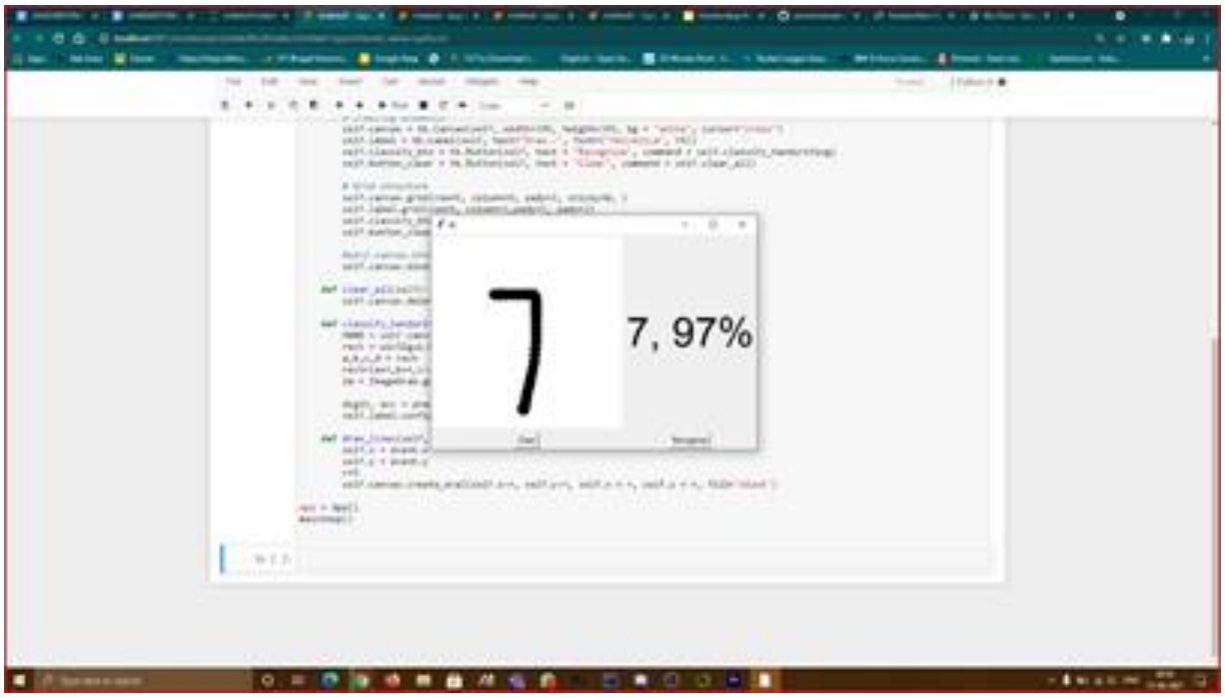
In Alphabet Recognition
•We first import modules TensorFlow, KERAS, NumPy, OpenCV, matplotlib, pandas, sklearn.
•Then we read the dataset which is downloaded locally in computer storage and print the dataset.
•Then we train and split the dataset in train and test.
•Then we will test the model in subplots.
•Then we read the download file to recognise the alphabet and predict the letter.
•The image is converted from RGB to grayscale.
•Then the final output is shown.

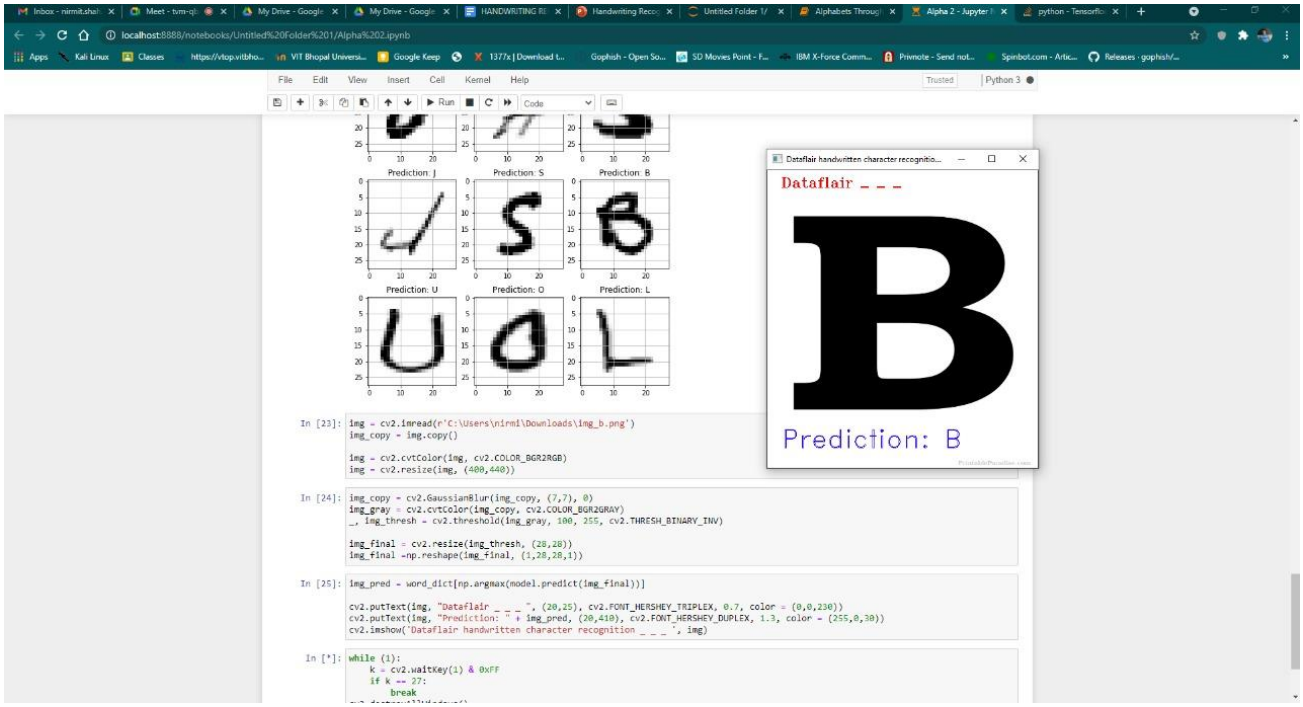
SNAP SHOTS OF THE PROJECT

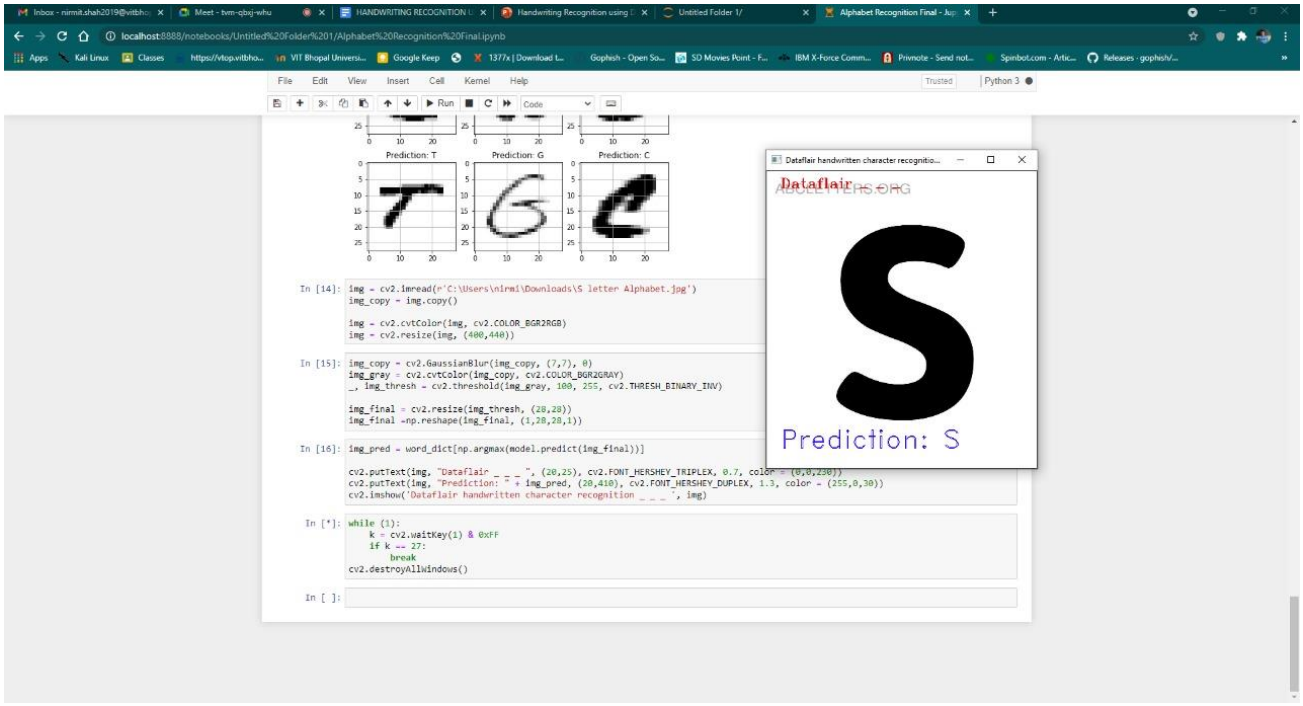



4, 33%


9, 58%


7, 97%


Prediction: J
Prediction: S
Prediction: B
Prediction: U
Prediction: O
Prediction: L
Dataflair _ _ _
B
Prediction: B
In [23]:
img = cv2.imread(r'C:\Users\nirm1\Downloads\img_b.png')
img_copy = img.copy()
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))
In [24]:
img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)
img_final = cv2.resize(img_thresh, (28,28))
img_final =np.reshape(img_final, (1,28,28,1))
In [25]:
img_pred = word_dict[np.argmax(model.predict(img_final))]
cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('Dataflair handwritten character recognition _ _ _ ', img)
In [*]:
while (1):
k = cv2.waitKey(1) & 0xFF
if k == 27:
break


Prediction: T
Prediction: G
Prediction: C
In [14]:
img = cv2.imread(r'C:\Users\nirmi\Downloads\S letter Alphabet.jpg')
img_copy = img.copy()
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))
In [15]:
img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)
img_final = cv2.resize(img_thresh, (28,28))
img_final =np.reshape(img_final, (1,28,28,1))
In [16]:
img_pred = word_dict[np.argmax(model.predict(img_final))]
cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('Dataflair handwritten character recognition _ _ _ ', img)
In [*]:
while (1):
k = cv2.waitKey(1) & 0xFF
if k == 27:
break
cv2.destroyAllWindows()
In [ ]:
Dataflair handwritten character recognitio...
Dataflair
Prediction: S

Implementation and Coding

For Digit Recognition

In [1]:
```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
```

In [2]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:
```
X_train = tf.keras.utils.normalize(X_train, axis=1)
X_test = tf.keras.utils.normalize(X_test, axis=1)
```

In [4]:
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))
model.add(tf.keras.layers.Dense(128, activation = tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation = tf.nn.softmax))
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
hist = model.fit(X_train, y_train, epochs=10)

val_loss, val_acc = model.evaluate(X_test, y_test)
print(val_loss, val_acc)
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4686 - accuracy: 0.8644
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1159 - accuracy: 0.9644
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0740 - accuracy: 0.9773
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0548 - accuracy: 0.9821
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0407 - accuracy: 0.9864
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0292 - accuracy: 0.9910
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0219 - accuracy: 0.9931
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0201 - accuracy: 0.9934
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0163 - accuracy: 0.9945
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0153 - accuracy: 0.9949
313/313 [==============================] - 1s 1ms/step - loss: 0.1171 - accuracy: 0.9724
0.11706514656543732 0.9724000096321106
```

In [5]:
```
model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

```
Saving the model as mnist.h5
```

In [6]:
```
from tensorflow.keras.models import load_model
from tkinter import *
import tkinter as tk
import win32gui
from PIL import ImageGrab, Image
import numpy as np
```

In [7]:
```
model = load_model('mnist.h5')

def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    #reshaping to support our model input and normalizing
    img = img.reshape(1,28,28,1)
    img = img/255.0
    #predicting the class
    res = model.predict([img])[0]
    return np.argmax(res), max(res)

class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

        self.x = self.y = 0

        # Creating elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")
        self.label = tk.Label(self, text="Draw..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command = self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Clear", command = self.clear_all)

        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)
```

```
        self.x = self.y = 0

        # Creating elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")
        self.label = tk.Label(self, text="Draw..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command = self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Clear", command = self.clear_all)

        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)

        #self.canvas.bind("<Motion>", self.start_pos)
        self.canvas.bind("<B1-Motion>", self.draw_lines)

    def clear_all(self):
        self.canvas.delete("all")

    def classify_handwriting(self):
        HWND = self.canvas.winfo_id()  # get the handle of the canvas
        rect = win32gui.GetWindowRect(HWND)  # get the coordinate of the canvas
        a,b,c,d = rect
        rect=(a+4,b+4,c-4,d-4)
        im = ImageGrab.grab(rect)

        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')

    def draw_lines(self, event):
        self.x = event.x
        self.y = event.y
        r=8
        self.canvas.create_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')

app = App()
mainloop()
```

For Alphabet Recognition

In [1]:
```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
```

In [2]:
```
data = pd.read_csv(r"C:\STUDY\Study\Project Exhibition-2\archive\A_Z Handwritten Data.csv").astype('float32')

print(data.head(10))
```

```
     0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  ...  0.639  0.640  0.641  \
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
5  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
6  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
7  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
8  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0
9  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ...    0.0    0.0    0.0

   0.642  0.643  0.644  0.645  0.646  0.647  0.648
0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5    0.0    0.0    0.0    0.0    0.0    0.0    0.0
6    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7    0.0    0.0    0.0    0.0    0.0    0.0    0.0
8    0.0    0.0    0.0    0.0    0.0    0.0    0.0
9    0.0    0.0    0.0    0.0    0.0    0.0    0.0

[10 rows x 785 columns]
```

In [3]:
```
X = data.drop('0',axis = 1)
y = data['0']
```

In [4]:
```
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
```

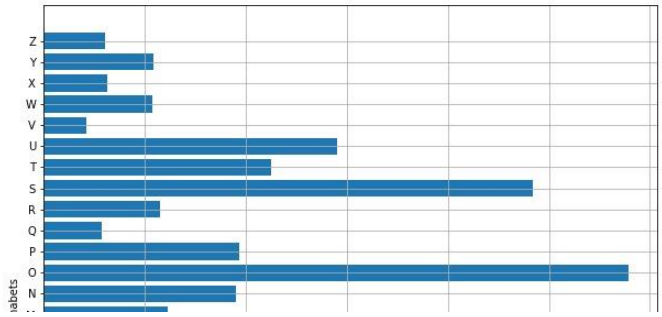
```
y = data['0']
```

In [4]:
```
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)

train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))

print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)
```

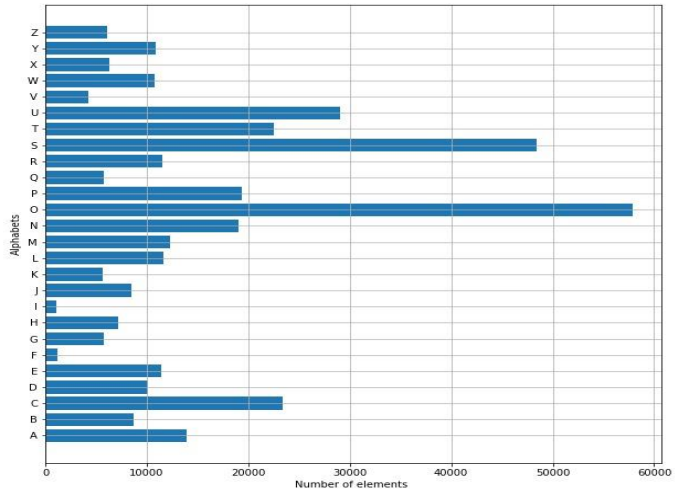
```
Train data shape:  (297960, 28, 28)
Test data shape:  (74490, 28, 28)
```

In [5]:
```
word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'N',14:'O',15:'P',16:'Q',17:'R',
```

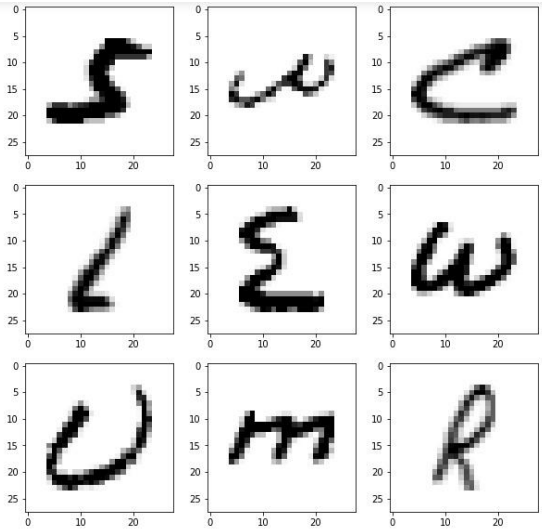
In [6]:
```
y_int = np.int0(y)
count = np.zeros(26, dtype='int')
for i in y_int:
    count[i] +=1

alphabets = []
for i in word_dict.values():
    alphabets.append(i)

fig, ax = plt.subplots(1,1, figsize=(10,10))
ax.barh(alphabets, count)

plt.xlabel("Number of elements ")
plt.ylabel("Alphabets")
plt.grid()
plt.show()
```




```
plt.xlabel("Number of elements ")
plt.ylabel("Alphabets")
plt.grid()
plt.show()
```



In [7]:
```
shuff = shuffle(train_x[:100])

fig, ax = plt.subplots(3,3, figsize = (10,10))
axes = ax.flatten()

for i in range(9):
    _, shu = cv2.threshold(shuff[i], 30, 200, cv2.THRESH_BINARY)
    axes[i].imshow(np.reshape(shuff[i], (28,28)), cmap="Greys")
plt.show()
```



In [8]:
```
train_X = train_x.reshape(train_x.shape[0],train_x.shape[1],train_x.shape[2],1)
print("New shape of train data: ", train_X.shape)

test_X = test_x.reshape(test_x.shape[0], test_x.shape[1], test_x.shape[2],1)
print("New shape of train data: ", test_X.shape)


#Now we reshape the train & test image dataset so that they can be put in the model.

#New shape of train data:  (297960, 28, 28, 1)
#New shape of train data:  (74490, 28, 28, 1)
```

```
New shape of train data:  (297960, 28, 28, 1)
New shape of train data:  (74490, 28, 28, 1)
```

In [9]:
```
train_yOHE = to_categorical(train_y, num_classes = 26, dtype='int')
print("New shape of train labels: ", train_yOHE.shape)

test_yOHE = to_categorical(test_y, num_classes = 26, dtype='int')
print("New shape of test labels: ", test_yOHE.shape)
```

```
New shape of train labels:  (297960, 26)
New shape of test labels:  (74490, 26)
```

In [10]:
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))

model.add(Dense(26,activation ="softmax"))
```

In [11]:
```
model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_X, train_yOHE, epochs=1,  validation_data = (test_X,test_yOHE))
```

```
Train on 297960 samples, validate on 74490 samples
297960/297960 [==============================] - 214s 719us/sample - loss: 0.1492 - accuracy: 0.9587 - val_loss: 0.0768 - val_a
ccuracy: 0.9788
```

In [12]:
```
print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])
```

```
The validation accuracy is : [0.9787891]
```

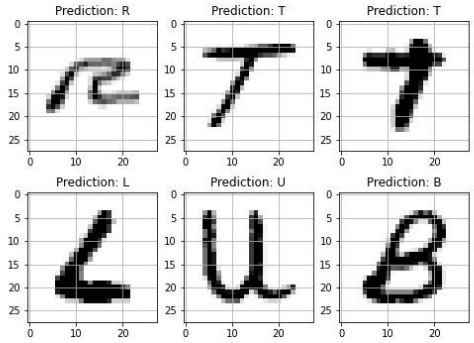
In [11]:
```
model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(train_X, train_yOHE, epochs=1,  validation_data = (test_X,test_yOHE))
```

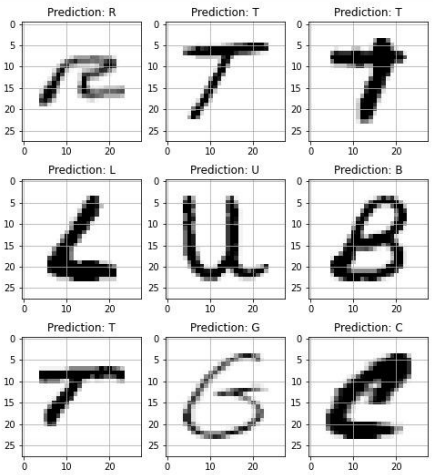
```
Train on 297960 samples, validate on 74490 samples
297960/297960 [==============================] - 214s 719us/sample - loss: 0.1492 - accuracy: 0.9587 - val_loss: 0.0768 - val_a
ccuracy: 0.9788
```

In [12]:
```
print("The validation accuracy is :", history.history['val_accuracy'])
print("The training accuracy is :", history.history['accuracy'])
print("The validation loss is :", history.history['val_loss'])
print("The training loss is :", history.history['loss'])
```

```
The validation accuracy is : [0.9787891]
The training accuracy is : [0.95866895]
The validation loss is : [0.07682637568060116]
The training loss is : [0.14916640240867526]
```

In [13]:
```
fig, axes = plt.subplots(3,3, figsize=(8,9))
axes = axes.flatten()

for i,ax in enumerate(axes):
    img = np.reshape(test_X[i], (28,28))
    ax.imshow(img, cmap="Greys")

    pred = word_dict[np.argmax(test_yOHE[i])]
    ax.set_title("Prediction: "+pred)
    ax.grid()
```





In [14]:
```
img = cv2.imread(r'C:\Users\nirmi\Downloads\S letter Alphabet.jpg')
img_copy = img.copy()

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))
```

In [15]:
```
img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

img_final = cv2.resize(img_thresh, (28,28))
img_final =np.reshape(img_final, (1,28,28,1))
```

In [16]:
```
img_pred = word_dict[np.argmax(model.predict(img_final))]

cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
```

0 10 20 0 10 20 0 10 20

In [14]:
```
img = cv2.imread(r'C:\Users\nirmi\Downloads\S letter Alphabet.jpg')
img_copy = img.copy()

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400,440))
```

In [15]:
```
img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)

img_final = cv2.resize(img_thresh, (28,28))
img_final =np.reshape(img_final, (1,28,28,1))
```

In [16]:
```
img_pred = word_dict[np.argmax(model.predict(img_final))]

cv2.putText(img, "Dataflair _ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('Dataflair handwritten character recognition _ _ _ ', img)
```

In [18]:
```
while (1):
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        break
cv2.destroyAllWindows()
```

Demo Video

Demo Video

CONCLUSION

We discussed a NN which is able to recognize text in images. The NN consists of 5 CNN and 2 RNN layers and outputs a character-probability matrix. This matrix is either used for CTC loss calculation or for CTC decoding. An implementation using TF is provided and some important parts of the code were presented.

REFERENCES

1. Abdul-Wahab,S.A., Al-Alawi,S.M. and El-Zawahry, Patterns of $S0_2$ emission: a refinery case study, *Environmental modeling & software*, 2002, 17, 563-570.

2. Aggarwal A.L, Sivacoumar R. and Goyal SK Air Quality Prediction : influence of model parameters and sensitivity analysis, *Indian Journal of Environmental Protection*, 1997, 17(9), 650-655.

LINKS

1. **Handwritten & Digit Recognition Using Deep Learning Technique**